Rapid Fourier space solution of linear partial integro-differential equations in toroidal magnetic confinement geometries.

B. F. McMillan[a], S. Jolliet[1a], A. Bottino[b], P. Angelino[2c], T. M. Tran[a], L. Villard[a]

[a] *Centre de Recherches en Physique des Plasmas, Association Euratom-Confédération Suisse, Ecole Polytechnique Fédérale de Lausanne, PPB, 1015 Lausanne, Switzerland.*
[b] *Max Planck Institut fur Plasmaphysik, IPP-EURATOM Association, Garching*
[c] *Association Euratom-CEA, CEA/DSM/DRFC Cadarache, France*

Abstract

Fluctuating quantities in magnetic confinement geometries often inherit a strong anisotropy along the field lines. One technique for describing these structures is the use of a certain set of Fourier components on the tori of nested flux surfaces. We describe an implementation of this approach for solving partial differential equations, like Poisson's equation, where a different set of Fourier components may be chosen on each surface according to the changing safety factor profile. Allowing the resolved components to change to follow the anisotropy significantly reduces the total number of degrees of freedom in the description. This can permit large gains in computational performance. We describe, in particular, how this approach can be applied to rapidly solve the gyrokinetic Poisson equation in a particle code, ORB5 [Jolliet *et. al.* Comp. Phys. Comm. 177, p409, (2007)], with a regular (non field-aligned) mesh.

The geometry of a toroidal magnetic confinement system with nested flux surfaces can be described using a toroidal coordinate system, with a radial coordinate s labelling each toroidal shell (or flux surface), and χ and ζ parameterising the poloidal and toroidal angle, respectively. We can choose the coordinates χ and ζ so that the magnetic field lines are straight, and are given by constant $q(s)\chi - \zeta$; q is called the safety factor.

Fourier descriptions of fluctuation quantities are particularly attractive for describing waves in toroidal magnetic confinement systems: many equilibrium[1], stability[2, 3, 4] and turbulence codes[5] use a Fourier description at some point. The Fourier description is natural because of the double periodicity of the flux surfaces in magnetic confinement devices. In practice, the radial direction tends

[1] Now at Japan Atomic Energy Agency, Higashi-Ueno-6-9-3, Taitou, Tokyo 110-0015, Japan
[2] Now at Institute of Chemical Sciences and Engineering, Ecole Polytechnique Fédérale de Lausanne, Switzerland

to be discretised using a finite-support scheme, and Fourier components are used to represent the variation along χ and ζ. As well as being natural, Fourier descriptions can also be quite efficient, due to the anisotropy of most of the waves of interest (e.g., low frequency Alfvén or drift waves) along the magnetic field lines. In a straight-field-line coordinate system, anisotropic waves can be described using a small set of Fourier components on each magnetic surface.

The usual Fourier representation of a field ϕ in magnetic devices is

$$\phi = \sum_j \sum_{(m,n)\in K} c_{j,(m,n)} \Lambda_j(s) \exp(im\chi - in\zeta), \tag{1}$$

where $\Lambda_j(s)$ are the finite element basis functions and K denotes the set of Fourier modes chosen, and $c_{j,(m,n)}$ are the discrete coefficients. The simple generalisation we suggest here is to allow K to vary with radial index j, so that the set of resolved Fourier modes varies across the radius. This is useful for anisotropic waves with small parallel wavenumber, $k_\| R \sim (nq(s) - m)B_\zeta/Bq(s) \lesssim 1$, because the relevant Fourier components (m, n) vary with radius (here R is the major radius and B_ζ/B is the toroidal component of the normalised magnetic field). In practise the amplitude of Fourier components (m, n) of these perturbations often decays rapidly to zero away from the resonant rational surface where $m = nq(s)$.

An alternative way to efficiently represent field-aligned waves is to use a field-aligned grid. This is particularly attractive for nonlinear codes, in which a spatial domain representation (or, equivalently, a convolution of spectral components) is necessary at some point. A field aligned grid has the advantage over a regular grid in χ and ζ that many fewer grid points are needed to represent field aligned waves. However, because of the varying and non-rational twist to magnetic field lines in a typical toroidal magnetic confinement geometry, it is quite complicated to implement a partial differential equation solver on a field-aligned grid (one implementation is described in ref. [6]). For example, it can be difficult to handle the magnetic axis using a field aligned grid.

The bulk of this paper deals with a particular application of the Fourier representation technique: the use of the Fourier representation for the Poisson solve in the gyrokinetic code ORB5[5]. ORB5 is a particle-in-cell gyrokinetic code using a control variates method, which can handle general axisymmetric global geometries as a background equilibria. In ORB5 a grid is chosen with similar node spacing in each direction, of size $N_\chi \times N_\zeta \times N_s$, where N_χ, N_ζ and N_s are the grid dimensions in the poloidal, toroidal and radial directions, respectively. A filter in Fourier space is used (instead of a field aligned grid) to ensure that only the waves of interest are resolved, in order to maximise the computational timestep and reduce sampling noise. The filter is applied to the perturbed ion density, and Poisson's equation is solved to determine the electric field in the plasma. Poisson's equation is discretised by representing the density and electric potential using (up to cubic) splines in radius and toroidal and poloidal angle.

The gyrokinetic Poisson equation is a three-dimensional partial integro-differential equation which is similar in structure to the true Poisson equation in

some limits, but which in general involves a mixture of integral and differential operators. The discrete form of the gyrokinetic Poisson equation is a matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{y} \tag{2}$$

for the gyro-density $\mathbf{y}$ in terms of the potential $\mathbf{x}$ which acts on the coefficients of the splines. Because the code solves in a axisymmetric geometry, the operator $\mathbf{A}$ is independent of the toroidal angle index, and a discrete Fourier transform is performed in toroidal angle which decouples modes of differing toroidal mode number n. In general the matrix A is different for each n, but the standard formulation of ORB5 the toroidal derivatives are ignored, and A is the same for each toroidal mode (with the possible exception of $n = 0$, which must be treated separately for the case of adiabatic electrons). For each n we have a set of spline coefficients labelled by (i, j) corresponding to the grid points on the cross section (s_i, χ_j). The matrix $\mathbf{A}$ is a band matrix (the interior of the band is in general not sparse when electrons are treated as adiabatic) and the inverse operation can be evaluated using standard numerical techniques. However, the matrix is potentially very large, and the Poisson solve can be slow and consume large amounts of memory. For ITG simulations, we need a grid resolution approximately equal to the ion sound gyroradius in each direction, and the total number of gridpoints in each direction is similar to $1/\rho^*$, where ρ^* is the ratio of the ion sound gyroradius to the minor radius. As a concrete example, in an ITG simulation of a reactor-scale plasma (like ITER), $\rho^* \sim 1/1000$, we would need approximately 10^3 grid points in each direction, and the code must solve 10^3 linear systems of rank 10^6 at each timestep. Solving these linear systems directly using a band solver (or, where possible, a sparse solver) became very costly in the ORB5 code even for considerably smaller grids, because these algorithms do not scale well.

Because of the field aligned filter, the number of degrees of freedom of the gyro-density vector is much smaller in Fourier space than in configuration space. Also, since the operator $\mathbf{A}$ is relatively narrow in Fourier space (the equilibrium and toroidicity terms are large only at long wavelength), the backsolved potential is also narrow in Fourier space. It therefore seems sensible to solve the matrix equation in Fourier space, discarding Fourier modes which are unimportant: that is, solving the Poisson equation in the subspace of field aligned modes.

We explain the algorithm for the Poisson solve in Fourier space, and why it leads to a large improvement in computational efficiency for our parallelised solver. We then present a more generic implementation of the Fourier solver using parallelised sparse matrices.

1. Fourier transforming the Poisson equation

One of the main computational steps in a PIC code is to rapidly determine the spatially continuous electric field and to deposit the charge at the marker positions. This is faster using a spline representation of the fields, rather than

a Fourier decomposition. We therefore choose to represent the gyrodensity and potential using a set of spline coefficients. A discrete Fourier transform on the spline coefficients is then used to rapidly solve a reduced matrix equation relating the two sets of spline coefficients. When all the Fourier modes are kept, the exact solution is the same as for the spatial domain solver: this provides a useful way check that the implementation is correct.

In Fourier space the RHS equation is written

$$y_F = FAF^{-1}x_F = A_F x_F \tag{3}$$

with F the Fourier transform, and $y_F = Fy$, $x_F = Fx$. We use complex, rather than real, Fourier transforms because this leads to a more elegant implementation. Because Hermiticity is conserved, exact conservation of energy is possible (in the zero-timestep and infinite number of markers limit) according to Ref. [7].

In ORB5, the Fourier space density filter is still applied to the density in Poisson's equation even when we solve the matrix equation in Fourier space: we can include more modes in the subspace for solving Poisson's equation than in the density filter if desired. We refer to the projection applied in Fourier space (to the matrix and vectors) to reduce the problem size as the *matrix filter*, to distinguish it from the density filter.

2. Details of Fourier components and poloidal indexing

For the Fourier solve, we construct a separate matrix for each toroidal Fourier mode: this means that n dependence of the matrix can be easily incorporated into the code, and simplifies parallelisation. For each toroidal mode we only need to keep weight/trial functions which are close to field aligned, with $m \in S_n = [nq(s_j) - \delta m, nq(s_j) + \delta m]$. The parameter δm, which is related to the maximum wavenumber resolved along the field line, is independent of j (the radial index) and n so that the matrix can be handled by standard band solvers. The number of poloidal modes kept for each surface is $1 + 2\delta m$. For algorithmic simplicity, it is useful to ensure that this interval of poloidal modes is inside the range $[-N_\chi/2, N_\chi/2 - 1]$. δm plays a similar role to the number of parallel grid points in a field-aligned description, restricting the maximum parallel wavenumber. We resolve a maximum wavenumber $k_\parallel \sim \delta m/Rq$. For a field-aligned grid, the maximum resolvable wavenumber $k_\parallel \sim N_\alpha/4Rq$, for N_α points along each poloidal turn of the field line. The typical choice of $N_\alpha \sim 16 - 32$, corresponds to a choice of $\delta m \sim 4 - 8$[8].

Because radial Dirichlet boundary conditions are implemented in our code using a rotation in the radial spline space, we also ensure that splines in the first and last few radial positions resolve the same range of poloidal modes: $m_{\min}(i) = m_{\min}(1)$ for $i \in [1, k]$ and $m_{\min}(i) = m_{\min}(N)$ for $i \in [N_s - k, N_s]$ (for kth order basis functions). In this case the mapping between poloidal array index and Fourier index are the same for these radial positions, and the rotation

in radial spline space (which is an independent rotation for each poloidal position or Fourier index) is the same as in the spatial domain solver.

The Poisson matrix in Fourier space can be constructed by explicitly Fourier transforming the configuration space matrix, but it is more memory and time efficient to perform the weak form integrations using the Fourier convolution theorem (as described in the appendix). Both construction methods are implemented in ORB5.

3. Computational implementation of the Poisson solve.

To solve the Poisson equation, we first take the toroidal and poloidal complex Fourier transform of the gyrodensity spline coefficients. At this step a filter is applied to the density to select physically relevant modes. We then project into the smaller space of field-aligned modes: this is a simple restriction operation, choosing the Fourier coefficients which are inside the matrix filter. The small Fourier space problem is solved using the (precomputed) Cholesky decomposition of the matrix, and the vector components are inserted into the full Fourier space. An inverse Fourier transform is then performed.

The purpose of projecting into this space is to reduce the size of the Poisson matrix, and speed up the calculation of the linear solution. Seen as a block matrix, the Fourier Poisson matrix has blocks of rank $2\delta m + 1$, instead of N_χ. This leads to a vast improvement in the scaling properties of the algorithm, because for the modes of interest, $k_\parallel R \sim 1$ leads to $\delta m \sim q$, which is fixed as the problem size increases, whereas $N_\chi \propto 1/\rho*$. For splines of order d, the number of super-diagonal blocks is d, and the number of non-zero elements in the matrix (and its Cholesky decomposition) and the memory use per matrix scales as $(d+1)(2\delta m+1)^2 N_s$ for the Fourier solver or $(d+1)N_\chi^2 N_s$ for the spatial domain solver. We typically have $N_\chi/\delta m \sim 1/\rho^*$ ($1/\rho^* = 100$ is common for production cases), so these matrices are much smaller than the spatial domain matrix. The code parallelization is a toroidal decomposition into P domains, and we need to store N_ζ/P of these matrices on every processor, but the memory usage is typically negligible compared to that used to store the spline coefficients in the spatial domain. For the spatial domain solver, memory usage can become a problem, even if the matrix is distributed across several processors.

The backsolve solution speed per toroidal mode is proportional to the number of non-zero elements in the matrix (the precomputation of the Cholesky decomposition is not usually burdensome), so the Fourier backsolve is much faster, often by several orders of magnitude: the backsolve typically takes negligible time compared to other parts of the code. The original backsolve became dominant in overall computational time and memory demand in ORB5 for large systems, so the speedup and memory reduction are valuable. For example, for $\rho^* = 1/140$, the standard spatial domain solver required 40% of the computational time, and around 100MB of memory per core. With the Fourier solver, the memory use is 134 times smaller, and the backsolve required 0.1% of the computational time. The difference becomes even more extreme for larger cases.

In fact, the improved matrix solver is usually much faster than the 2D Fourier transform, which involves a parallel transpose in our parallelisation scheme. For sufficiently large problems (around grid sizes of $N_s \times N_{chi} \times N_\zeta = 1024 \times 2048 \times 2048$) the number of spatial grid points (which scales like the system volume) starts to become comparable to the number of computational markers used in the PIC scheme (which scales like the cross-sectional area of the device). The consequence is that grid-based operations dominate and that memory use and the computer time per timestep start to scale like the inverse cube of ρ^* rather than $1/\rho^{*2}$. For ITG simulations, the scaling starts to become non-optimal at a plasma size of about $1/\rho* \sim 1000$, around the size of ITER, which is likely to be the largest fusion device built over the coming decades. For smaller systems, resolving the dynamics in the 5-D space using the markers dominates the computation, despite the redundancy of representing anisotropic variation on an isotropic grid.

As an aside, when the spatial domain matrix is sparse (this only occurs in specialised cases where there is no adiabatic electron species), using an iterative sparse solver is a possible alternative way to reduce storage requirements. We had earlier implemented an iterative sparse solver in the spatial domain, which allowed an increase in grid size over the standard solver in the spatial domain: using the sparse solver, the maximum feasible grid size was $512 \times 2048 \times 1024$, corresponding to $\rho^* \sim 1/560$. For this case using the Fourier solver rather than the sparse solver allows a threefold reduction in overall computational time per timestep and halving of total memory use. This is a consequence of the better scaling of the Fourier technique with system size.

4. Using the Fourier solver in stellarator geometry and more general problems

The approach as detailed earlier was restricted in certain ways: it becomes inefficient for large q, and in particular, shows no improvement over a direct solution in cases where $q \to \infty$, because δm should also generally be proportional to q in order to handle field-aligned variations. Also, the analysis was restricted to axisymmetric background geometries where a continuous rotation symmetry was present. For problems in non-axisymmetric geometry, such as arise in stellarator physics, toroidal coupling prevents the separation of the problem in toroidal Fourier index, so that all the Fourier modes are coupled together. In Fourier space, the problem is not strictly sparse, but the terms rapidly become small far from the diagonal, and can be ignored or treated as a correction. This implies that a sparse solver in Fourier space might be an appropriate method to solve the matrix problem.

As a proof of principle, we implemented a finite element model (FEM) solver which uses the transform into discrete Fourier space to construct the matrix problem, and used a parallel sparse solver, PETSc[9], to perform a backsolve. Using a sparse solver (as opposed to a banded solver) allows us to store only the coupling terms with significant strength, so that the matrix storage is not excessive even for fairly large problems. This is useful, for example, near the

plasma edge, where (for diverted tokamaks) $q \to \infty$ and a wide range of poloidal modes are nearly field-aligned at a single n.

The algorithm involves two indexing schemes, $i \in [1, N_m]$ for the full Fourier space and $i' \in [1, N_k]$ for the restricted Fourier space. The restriction and expansion operations are very simple to program once subroutines have been built to find the mapping $i' \to i$. Matrix construction is slightly more complicated, because of the amount of indexing which burdens the code. A matrix construction routine generally loops over weight function indexes $i' \in N_r$. For each i', the coupled trial function indexes j' need to be found. One efficient way to perform this operation is to look up the full index, i, then find the radial index x_i and Fourier index (m_i, n_i). We then loop over all the modes with $s_j \in s_i \pm \delta s$, $m_j \in m_i \pm \delta m(s_i)$ and $n_j \in n_i \pm \delta n(s_i)$ and add a coefficient to the matrix if it is in the restricted space. This is exactly analogous to the process needed in finite element codes with unstructured grids, where a graph of coupled nodes needs to be constructed. In practice it is often simpler or faster to set up the storage of the matrix first, by performing an initial loop over the trial and weight functions, and inserting the coefficients later (perhaps in an arbitrary order). Standard sparse matrix packages allow simple implementation of this two-step construction technique.

A moderate size test case was run on 8 processors resolving relevant Fourier modes on a $1024 \times 1024 \times 512$ grid. The physical geometry of the grid is an oblate cylinder (the cross-section is an ellipse with aspect ratio of 2), with a sinusoidal modulation to the z coordinate along the axis of the ellipse ($z' \to z + Asin(z)$ produces a coupling between the Fourier modes in z). The Poisson equation $\nabla^2 V = U$ was represented using linear finite elements. We restricted the modes to $|m - nq| < \epsilon$, with $q(r) = 1.2 + 0.5r$, and $\epsilon = 7$. Only coupling between modes $m_i - m_j \leq 5$ and $n_i - n_j \leq 2$ was included in the matrix. For this system, we resolve 3840 resonant modes per surface. For 8 Processors, 1.7 Gigabytes of memory was used in matrix construction, and the backsolve required 200 seconds to achieve a relative accuracy of 1×10^{-5}. For comparison, if we resolved each resonant Fourier component over the entire radial extent, we would require ~ 50 times as many degrees of freedom in the matrix, and the matrix would be proportionally larger. Solving the problem in the spatial domain would require more than 200 times as many degrees of freedom; even if only nearest neighbour coupling coefficients were stored, 115 gigabytes would be needed just for the sparse matrix storage, before any inversion could be attempted.

5. Conclusions

The Fourier-transform solver described here allowed a massive speedup and reduction in memory resources in the spatial part of our gyrokinetic code. This kind of semi-spectral technique has been applied before: the novel ingredient is that we choose a different set of Fourier components at each surface. This method appears to be generally applicable in codes which treat waves aligned with the magnetic fields of confinement devices and should allow performance improvements to codes which currently resolve a fixed set of Fourier components

across the entire minor radius. The code changes required to implement such a solver are rather superficial because all we have performed is a projection of the problem onto a smaller space; the main burden is to provide a mapping between the full and restricted space and a means to determine which components are coupled together.

References

[1] S. P. Hirshman, O. Betancourt, Preconditioned descent algorithm for rapid calculations of magnetohydrodynamic equilibria, J. Comput. Phys. 96 (1) (1991) 99–109. doi:http://dx.doi.org/10.1016/0021-9991(91)90267-O.

[2] D. V. Anderson, W. A. Cooper, R. Gruber, S. Merazzi, U. Schwenn, The Terpsichore Code for the Stability Analysis of Magnetically Confined Fusion Plasmas, Supercomputer 8 (3) (1991) 32–35.

[3] B. F. McMillan, R. G. Storer, Spector3d: a resistive magnetohydrodynamic stability code for stellarators, Journal of Plasma Physics 72 (2006) 829–832.

[4] P. Popovich, W. Cooper, L. Villard, A full-wave solver of the Maxwell's equations in 3d cold plasmas, Computer Physics Communications 175 (4) (2006) 250 – 263.

[5] S. Jolliet, A. Bottino, P. Angelino, R. Hatzky, T. Tran, B. McMillan, O. Sauter, K. Appert, Y. Idomura, L. Villard, A global collisionless PIC code in magnetic coordinates, Computer Physics Communications 177 (2007) 409–425.

[6] Y. Nishimura, Z. Lin, J. Lewandowski, S. Ethier, A finite element Poisson solver for gyrokinetic particle simulations in a global field aligned mesh, Journal of Computational Physics 214 (2006) 657–671.

[7] R. Hatzky, T. M. Tran, A. Könies, R. Kleiber, Energy conservation in a nonlinear gyrokinetic particle-in-cell code for ion-temperature-gradient-driven modes in theta-pinch geometry, Physics of Plasmas 9 (2002) 898.

[8] S. Jolliet, Gyrokinetic particle-in-cell global simulations of ion-temperature-gradient and collisionless-trapped-electron-mode turbulence in tokamaks., Ph.D. thesis, EPFL (2009).

[9] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.0.0, Argonne National Laboratory (2008).

A. Convolution construction of the Fourier transformed Poisson matrix

We consider here the discretisation of the gyrokinetic Poisson equation in the long wavelength limit. Typical terms in the matrix $A_{(kj)(k'j')}$, with k and k' the weight and trial function radial indexes, and j and j' the poloidal indexes, can be written as a sum of n_w terms, each term being a product of the weight and trial functions ($\Lambda_{k,j}$ and $\Lambda_{k',j'}$ or their derivatives) and a spatially varying equilibrium function $C_w(s,\chi)$. There is also a nonlocal 'zonal flow' term (a flux surface average) due to the modelling of the adiabatic electrons which must be handled differently. In the standard case the integrations are approximated in the code via a sum over q Gauss quadrature points and N_s intervals in radius, N_χ intervals in poloidal position as

$$A_{(kj)(k'j')} = \sum_{w=1}^{n_w} \sum_{q=1}^{n_q} \sum_{K=0}^{N_s-1} \sum_{J=0}^{N_\chi-1} \Lambda_k^{u_w}(s)\Lambda_j^{v_w}(\chi) C_w(s,\chi) \Lambda_{k'}^{u_w}(s)\Lambda_{j'}^{v_w}(\chi)\Big|_{(s,\chi)=(s_{K,q},\chi_{J,q})} \tag{4}$$

Here, n_w is the number of terms in the weak form and n_q is the number of quadrature points, and u_w and v_w describe the number of times the spline functions must be differentiated for each term in the weak form. The evaluation position (s,χ) depends on the quadrature point q and the interval indices K and J. The periodicity of the poloidal coordinate χ is implicit. Note that many terms will be zero (compact support of spline functions). The Fourier transform of $A_{(kj)(k'j')}$ is written

$$A'_{(km)(k'm')} = \frac{1}{N_\chi} \sum_{j=0}^{N_\chi-1} \sum_{j'=0}^{N_\chi-1} \exp\left[-\frac{2\pi i}{N_\chi}(mj - m'j')\right] A_{(kj)(k'j')} \tag{5}$$

We now show how this can be written in a convolution form. The sums over the radius, weak form and quadrature points can be moved outside of the Fourier transform, and we only need consider the poloidal sums over the interval: we define the summand B via

$$A'_{(km)(k'm')} = \frac{1}{N_\chi} \sum_{w=1}^{n_w} \sum_{q=1}^{n_q} \sum_{K=0}^{N_s-1} \Lambda_k^{u_w}(s)\Lambda_{k'}^{u_w}(s)|_{s=s_{K,q}} B'_{(m)(m')qwK}. \tag{6}$$

The poloidal positions of the quadrature points can be written $\chi_{J,q} = \chi_q + 2\pi J/N_\chi$. And we have

$$B'_{(m)(m')qwK} = \sum_{j=0}^{N_\chi-1} \sum_{j'=0}^{N_\chi-1} \sum_{J} \exp\left[-\frac{2\pi i}{N_\chi}(mj - m'j')\right] \Lambda_j^{v_w}(\chi_q + 2\pi J/N_\chi) \Lambda_{j'}^{v_w}(\chi_q + 2\pi J/N_\chi) C_w(s_{K,q}, \chi_q + 2\pi J/N_\chi). \tag{7}$$

We also have $\Lambda_j(\chi) = \Lambda_0(\chi - 2\pi j/N_\chi)$ (because the spline functions are all identical displaced copies), so we can write:

$$B'_{(m)(m')qwK} = \sum_{j=0}^{N_\chi-1} \sum_{j'=0}^{N_\chi-1} \sum_{J} \exp\left[-\frac{2\pi i}{N_\chi}(mj - m'j')\right] \Lambda_0^{v_w}(\chi_q + 2\pi(J-j)/N_\chi) \Lambda_0^{v_w}(\chi_q + 2\pi(J-j')/N_\chi) C_w(s_{K,q}, \chi_q + 2\pi J/N_\chi). \tag{8}$$

The sum indices can be cyclically permuted and the sums separated to give

$$\begin{aligned} B'_{(m)(m')qwK} &= \sum_{j=0}^{N_\chi-1} \Lambda_j^{v_w}(\chi_q) \exp[-2\pi i m j/N_\chi] \sum_{j'=0}^{N_\chi-1} \Lambda_{j'}^{v_w}(\chi_q) \exp[2\pi i m' j'/N_\chi] \\ &\quad \sum_J C_w(s_{K,q}, \chi_q + 2\pi J/N_\chi) \exp[2\pi i (m-m')J/N_\chi] \\ &= Z_{\mathbf{P}}(m) Z_{\mathbf{P}}(m') C'_w(s_{K,q}, m - m'), \end{aligned} \tag{9}$$

which is a product of three separate 1-D Fourier transforms in j,j' and J. After performing these transforms, the terms $B'_{(im)(i'm')qwI}$ can be rapidly evaluated.

The zonal flow operator involves a surface average. This leads to terms in the weak form which involve a product of sums, $B^{ZF}_{(j)(j')qwK} = P_{(j)qwK} P_{(j')qwK}$, with

$$P_{(j)qwK} = \sum_{q=1}^{n_q} \sum_{J=0}^{M} G(s_{K,q}, \chi_{J,q}) \Lambda_j(\chi_{J,q}), \tag{10}$$

where G is a geometrical term. The Fourier transform of B^{ZF} can be straightforwardly expressed as $B'^{ZF}_{(im)(i'm')} = P'_{(m)qwK} P'_{(m')qwK}$ in terms of Fourier transforms P'_m of P_j.